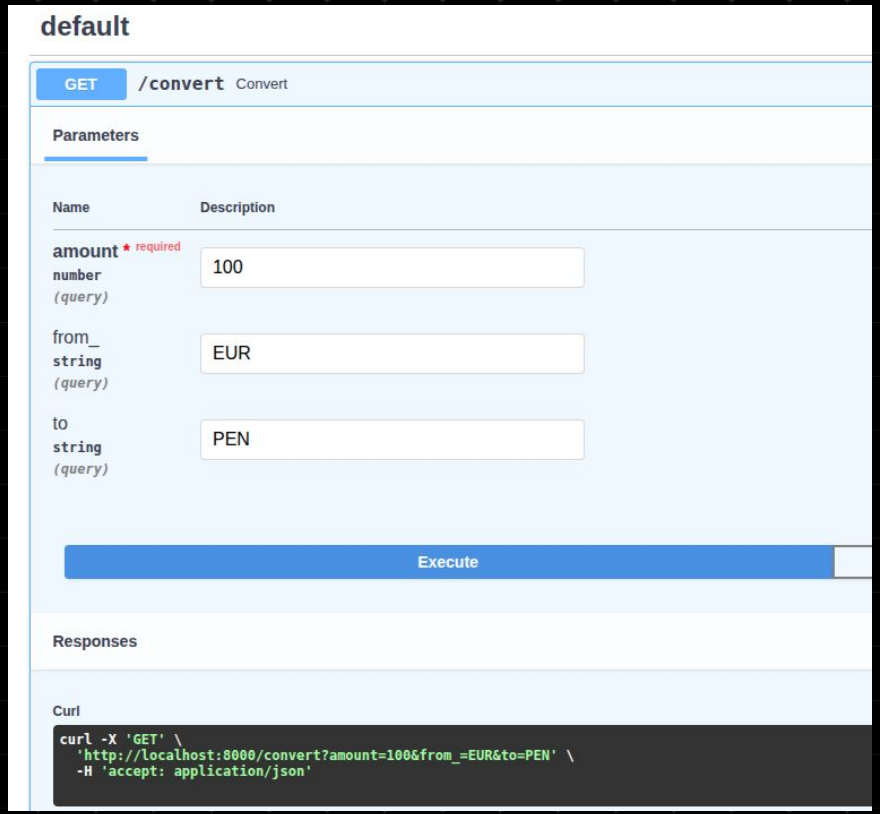
PC 1 - Calidad de Software

Integrantes:

Piero Alexis Violeta Estrella — 20184065H
Andrés Sebastián La Torre Vasquez — 20212100C
Maxwel Paredes Lopez — 20191179E
Franklin Espinoza Pari — 20210135D
Arturo Hinostroza Olivera — 20191548K

Implementamos un endpoint REST con FastAPI.

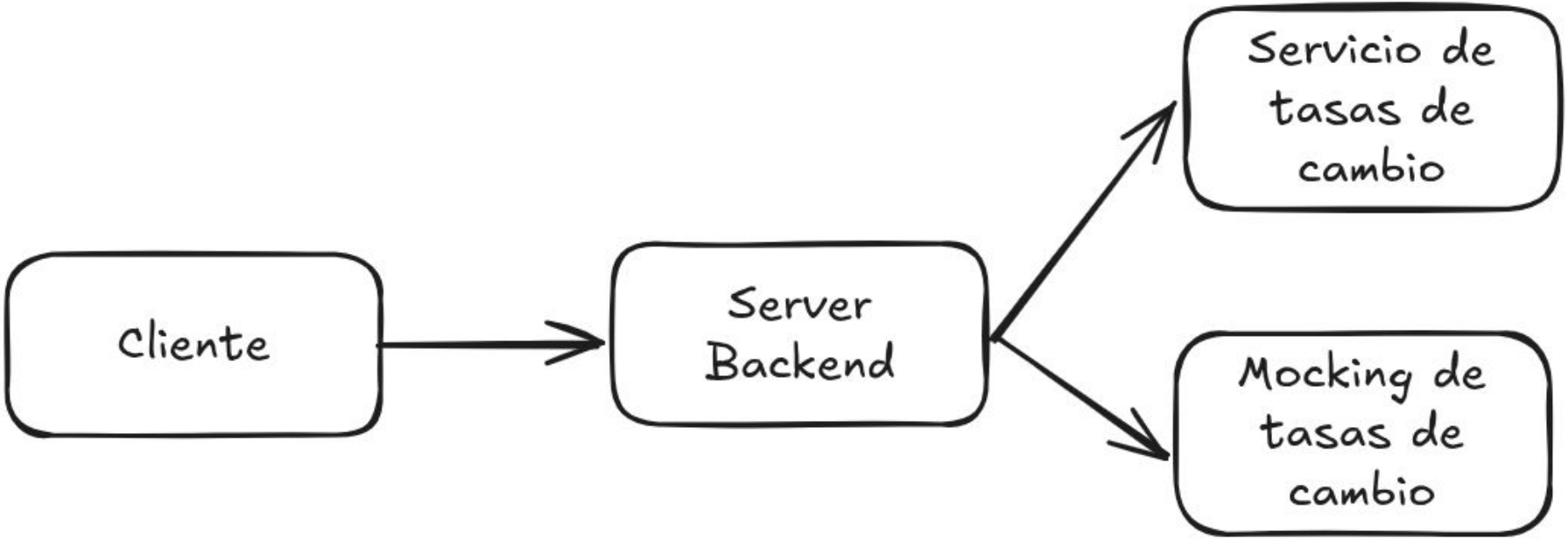


Entrada: amount, from_, to.

Salida: monto convertido con la tasa de cambio.

La tasa proviene de un servicio externo (pero lo mockearemos en tests/demos).

Problema : Dependemos de un servicio externo (en nuestro caso api de tasas de cambio).



Riesgos : puede estar caído, lento o costoso de usar.

Solución : Mocks / Fakes para aislar nuestra lógica de negocio y poder testear.

Diseño con Puertos y Adaptadores

```
// class RatesPort(Protocol):
    async def get_rate(self, from_: str, to: str) → float:
        """Return the FX rate to convert from → to."""
        ...
        # SERVICIO SIMULADO COMO NO DISPONIBLE
```

Lógica de Negocio

```python
// class ConversionService:
    def __init__(self, rates: RatesPort):
        self.rates = rates

    async def convert(self, amount: float, from_: str, to: str) -> dict:
        if amount <= 0:
            raise ValueError("amount must be > 0")
        if from_ == to:
            raise ValueError("from and to must differ")
        rate = await self.rates.get_rate(from_, to)
        converted = round(amount * rate, 2)
        return {
            "amount": amount,
            "from": from_,
            "to": to,
            "rate": rate,
            "converted": converted,
        }
```

Adaptador Mockeado

```python
class FakeRatesAdapter(RatesPort):
    def __init__(self, table: Dict[Tuple[str, str], float] | None = None):
        self.table = table or {
            ("USD", "PEN"): 3.72,
            ("EUR", "PEN"): 4.05,
            ("PEN", "USD"): 0.27,
        }

    async def get_rate(self, from_: str, to: str) -> float:
        try:
            return self.table[(from_, to)]
        except KeyError as exc:
            raise LookupError("currency pair not supported") from exc
```

FastAPI Endpoint

```python
@router.get("/convert", response_model=ConvertResponse)
async def convert(
    amount: float = Query(..., gt=0),
    from_: str = Query("USD"),
    to: str = Query("PEN"),
    rates: RatesPort = Depends(get_rates_port),
):
    svc = ConversionService(rates)
    try:
        result = await svc.convert(amount, from_, to)
        return {**result, "from_": result["from"], "source": "fake"}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except LookupError as e:
        raise HTTPException(status_code=400, detail=str(e))
```

Mocking en Tests

```python
@pytest.mark.asyncio
async def test_convert_ok():
    """
    GIVEN   a ConversionService with a mocked RatesPort returning 3.5
    WHEN    converting 100 USD to PEN
    THEN    the result should include rate=3.5 and converted=350.0
    """
    rates = AsyncMock()
    rates.get_rate.return_value = 3.5
    svc = ConversionService(rates)
    out = await svc.convert(100, "USD", "PEN")
    assert out["rate"] == 3.5
    assert out["converted"] == 350.0
```

Mocking en Tests

```python
@pytest.mark.asyncio
async def test_convert_invalid_amount():
    """
    GIVEN   a ConversionService with a mocked RatesPort
    WHEN    converting with amount=0
    THEN    a ValueError should be raised
    """
    rates = AsyncMock()
    svc = ConversionService(rates)
    with pytest.raises(ValueError):
        await svc.convert(0, "USD", "PEN")


@pytest.mark.asyncio
async def test_convert_same_currency():
    """
    GIVEN   a ConversionService with a mocked RatesPort
    WHEN    converting from and to the same currency (USD -> USD)
    THEN    a ValueError should be raised
    """
    rates = AsyncMock()
    svc = ConversionService(rates)
    with pytest.raises(ValueError):
        await svc.convert(10, "USD", "USD")
```

Validando que los tests corran bien

```
pierov@pierov-Inspiron-15-3520:~/Piero/UNI-courses/SoftwareQuality/pc1-mocking-demo$ make test
poetry run pytest -q
.....
5 passed in 0.36s
```